

合肥工业大学

操作系统实验报告

实验题目	实验 2 操作系统的启动
学生姓名	袁焕发
学 号	2019217769
专业班级	物联网工程 19-2 班
指导教师	田卫东
完成日期	2021 年 11 月 2 日

1. 实验目的和任务要求

跟踪调试 EOS 在 PC 机上从加电复位到成功启动的全过程，了解操作系统的启动过程。

查看 EOS 启动后的状态和行为，理解操作系统启动后的工作方式。

2. 实验原理

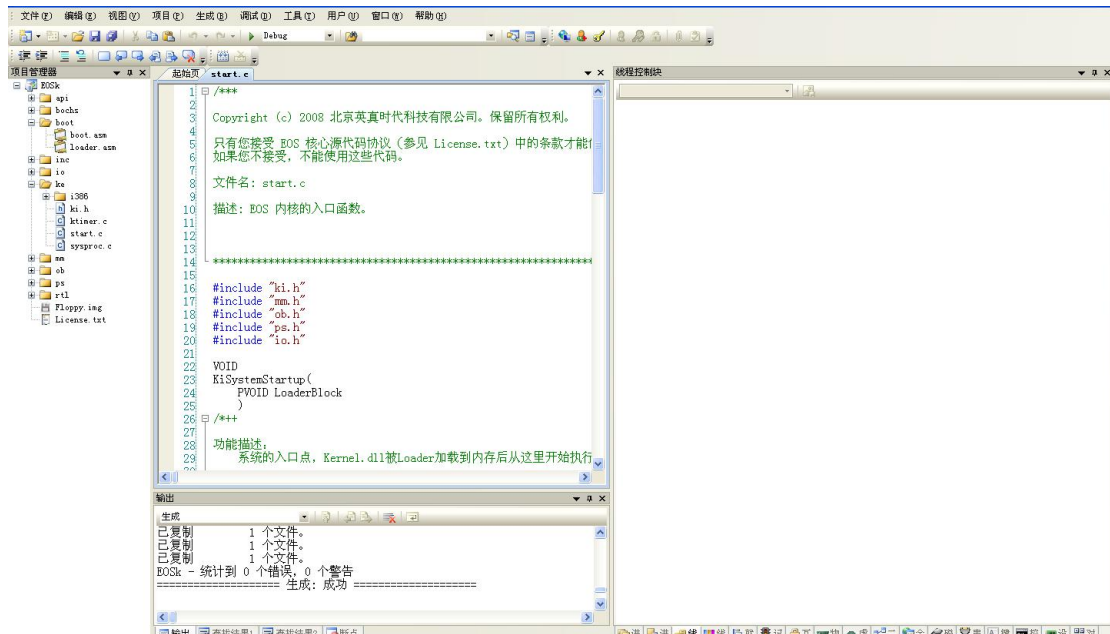
EOS 是一个可以在 Intel X86 平台上运行的、面向教学的开源操作系统，有配套的 IDE 实验环境，可以将 EOS 源代码编译为二进制文件（包括引导程序和内核）或者将编译好的二进制文件写入一个软盘镜像（或软盘），然后让虚拟机（或裸机）运行此软盘中的 EOS，并对其进行远程调试在。IDE 环境成功生成 EOS 的二进制文件后，会自动生成 EOS SDK，在获得 SDK 之后就可以根据其提供的 API 进行程序的编写。

EOS 启动时, BIOS 执行开机自检和初始化, 将软盘引导扇区加载到物理内存的 0x7C00 处, 并跳转到引导扇区的 Boot 程序中执行; Boot 程序将软盘根目录中的 Loader 程序 Loader.bin 文件加载到物理内存的 0x1000 处, 并跳转到 Loader 程序中执行; Loader 程序将软盘根目录中的操作系统内核 Kernel.d11 文件加载到物理内存中, 然后启动 CPU 的保护模式和分页机制, 最后跳转到 Kernel.d11 的入口点函数中执行; EOS 内核完成初始化后, 用户即可与之进行交互。

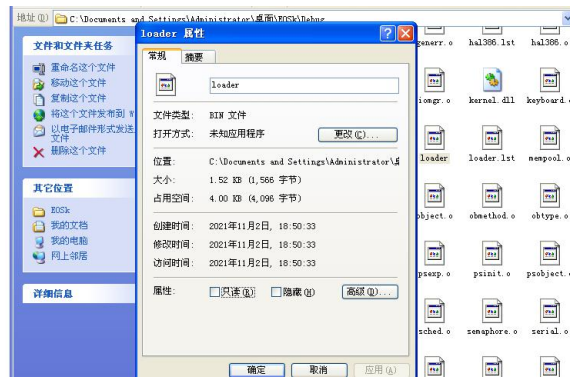
3. 实验内容

3.1. 准备实验

启动 OS Lab，新建一个 EOS Kernel 项目，并生成项目。



找到 loader.asm 生成的加载程序 loader.bin 文件，记录文件的大小 1566 字节。



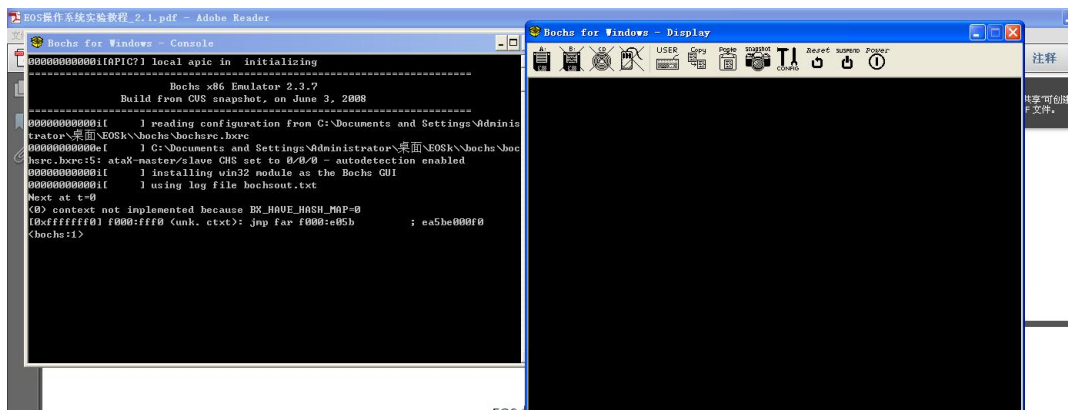
3.2. 调试 EOS 操作系统的启动过程

3.2.1. 使用 Bochs Debug 做为远程目标机

打开“项目管理器”窗口中的“属性”，在弹出的“属性页”列表中找到“远程目标机”属性，将调试时使用的远程目标机修改为 Bochs Debug，并调试程序。

3.2.2. 调试 BIOS 程序

启动调试后，Bochs 会在 CPU 要执行的第一条指令（即 BIOS 的第一条指令）处中断。此时，Display 窗口还没有显示任何内容，Console 窗口会显示将要执行的 BIOS 第一条指令的相关信息，并等待用户输入调试命令。



查看 CPU 在没有执行任何指令之前主要寄存器和内存中的数据。

窗口中输入调试命令 sreg 后按回车，显示当前 CPU 中各个段寄存器的值，输入调试命令 r 后按回车，显示当前 CPU 中各个通用寄存器的值。

```
Bochs for Windows - Console
[0xffffffff] f000:fff0 <unk. ctxt>: jmp far f000:e05b ; ea5be000f0
<bochs:1> sreg
cs:s=0xf000, dl=0x0000ffff, dh=0xff0093ff, valid=1
ds:s=0x0000, dl=0x0000ffff, dh=0x00009300, valid=1
ss:s=0x0000, dl=0x0000ffff, dh=0x00009300, valid=1
es:s=0x0000, dl=0x0000ffff, dh=0x00009300, valid=1
fs:s=0x0000, dl=0x0000ffff, dh=0x00009300, valid=1
gs:s=0x0000, dl=0x0000ffff, dh=0x00009300, valid=1
ldtr:s=0x0000, dl=0x0000ffff, dh=0x00008200, valid=1
tr:s=0x0000, dl=0x0000ffff, dh=0x00008b00, valid=1
gdttr:base=0x00000000, limit=0xffff
idtr:base=0x00000000, limit=0xffff
<bochs:2> r
rax: 0x00000000:00000000 rcx: 0x00000000:00000000
rdx: 0x00000000:00000f20 rbx: 0x00000000:00000000
rsp: 0x00000000:00000000 rbp: 0x00000000:00000000
rsi: 0x00000000:00000000 rdi: 0x00000000:00000000
r8 : 0x00000000:00000000 r9 : 0x00000000:00000000
r10: 0x00000000:00000000 r11: 0x00000000:00000000
r12: 0x00000000:00000000 r13: 0x00000000:00000000
r14: 0x00000000:00000000 r15: 0x00000000:00000000
rip: 0x00000000:0000ffff
eflags 0x00000002
id vip vif ac vm rf nt IOPL=0 of df if tf sf zf af pf cf
<bochs:3>
```

输入调试命令 xp /1024b 0x0000, 查看开始的 1024 个字节的物理内存
输入调试命令 xp /512b 0x7c00, 查看软盘引导扇区应被加载到的内存位置。

```
EOS操作系统实验教程_2.1.pdf - Adobe Reader
Bochs for Windows - Console
0x000000000000003a0 <bogus+ 928>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000003a8 <bogus+ 936>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000003b0 <bogus+ 944>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000003b8 <bogus+ 952>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000003c0 <bogus+ 960>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000003c8 <bogus+ 968>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000003d0 <bogus+ 976>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000003d8 <bogus+ 984>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000003e0 <bogus+ 992>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000003e8 <bogus+ 1000>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000003f0 <bogus+ 1008>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000003f8 <bogus+ 1016>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
<bochs:4>
```

```
EOS操作系统实验教程_2.1.pdf - Adobe Reader
Bochs for Windows - Console
0x0000000000007da0 <bogus+ 416>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007da8 <bogus+ 424>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007db0 <bogus+ 432>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007db8 <bogus+ 440>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007dc0 <bogus+ 448>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007dc8 <bogus+ 456>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007dd0 <bogus+ 464>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007dd8 <bogus+ 472>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007de0 <bogus+ 480>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007de8 <bogus+ 488>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007df0 <bogus+ 496>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007df8 <bogus+ 504>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
<bochs:5>
```

3.2.3. 调试软盘引导扇区程序

输入调试命令 `vb 0x0000:0x7c00`，在逻辑地址 `0x0000:0x7c00` 处添加了一个断点。输入调试命令 `c` 继续执行，在 `0x7c00` 处中断。中断后会在 Console 窗口中输出下一个要执行的指令，即软盘引导扇区程序的第一条指令。

```
CPU 0: HALTED
CPU 0: HALTED
CPU 0: HALTED
CPU 0: HALTED
CPU 0: HALTED
<0> Breakpoint 4619281, in 0000:7c00 <0x00007c00>
Next at t=16897706
<0> [0x00007c00] 0000:7c00 <unk. ctxt>: jmp .+0x006d <0x00007c6f>; eb6d
<bochs:7>
```

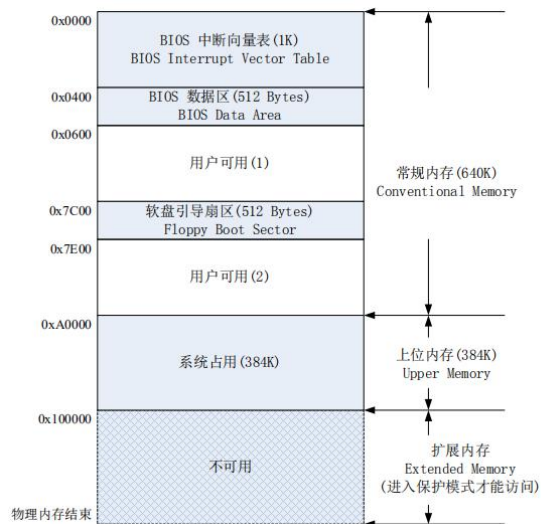
输入调试命令 `sreg` 验证 CS 寄存器 (`0x0000`) 的值，输入调试命令 `r` 验证 IP 寄存器 (`0x7c00`) 的值。

```
EOS操作系统实验教程_2.1.pdf - Adobe Reader
Bochs for Windows - Console
<0> [0x00007c00] 0000:7c00 <unk. ctxt>: jmp .+0x006d <0x00007c6f>; eb6d
<bochs:7> sreg
cs:s=0x0000, dl=0x0000ffff, dh=0x00009300, valid=1
ds:s=0x0000, dl=0x0000ffff, dh=0x00009300, valid=1
ss:s=0x0000, dl=0x0000ffff, dh=0x00009300, valid=7
es:s=0x0000, dl=0x0000ffff, dh=0x00009300, valid=1
fs:s=0x0000, dl=0x0000ffff, dh=0x00009300, valid=1
gs:s=0x0000, dl=0x0000ffff, dh=0x00009300, valid=1
ldtr:s=0x0000, dl=0x0000ffff, dh=0x00008200, valid=1
tr:s=0x0000, dl=0x0000ffff, dh=0x00008b00, valid=1
gdt:base=0x000fb632, limit=0x30
idt:base=0x00000000, limit=0x3ff
<bochs:8> r
rax: 0x00000000:0000aa55 rcx: 0x00000000:00000000
rdx: 0x00000000:00000000 rbx: 0x00000000:00000000
rsp: 0x00000000:0000ffda rbp: 0x00000000:00000000
rsi: 0x00000000:ffff0000 rdi: 0x00000000:0000ffac
r8 : 0x00000000:00000000 r9 : 0x00000000:00000000
r10: 0x00000000:00000000 r11: 0x00000000:00000000
r12: 0x00000000:00000000 r13: 0x00000000:00000000
r14: 0x00000000:00000000 r15: 0x00000000:00000000
rip: 0x00000000:00007c00
eflags 0x00000082
id vip vif ac vm rf nt IOPL=0 of df if tf SF zf af pf cf
<bochs:9>
```

输入调试命令 `xp /1024b 0x0000` 验证此时 BIOS 中断向量表已经被载入，输入调试命令 `xp /512b 0x7c00` 显示软盘引导扇区程序的所有字节码。

```
EOS操作系统实验教程_2.1.pdf - Adobe Reader
Bochs for Windows - Console
0x0000000000007da0 <hogus+ 416>: 0x18 0x7c 0xf6 0xf3 0xfe
0xc4 0x88 0xe1
0x0000000000007da8 <hogus+ 424>: 0x88 0xc6 0xd0 0xe8 0x88
0xc5 0x80 0xe6
0x0000000000007db0 <hogus+ 432>: 0x01 0x8a 0x16 0x24 0x7c
0x5b 0xb4 0x02
0x0000000000007db8 <hogus+ 440>: 0x8a 0x46 0xfe 0xcd 0x13
0x0f 0x82 0xf5
0x0000000000007dc0 <hogus+ 448>: 0xff 0x59 0x5d 0xc3 0x00
0x00 0x00 0x00
0x0000000000007dc8 <hogus+ 456>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007dd0 <hogus+ 464>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007dd8 <hogus+ 472>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007de0 <hogus+ 480>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007de8 <hogus+ 488>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007df0 <hogus+ 496>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007df8 <hogus+ 504>: 0x00 0x00 0x00 0x00 0x00
0x00 0x55 0xaa
<bochs:11>
```

输入调试命令 `xp /512b 0x0600` 验证第一个用户可用区域是空白的。
输入调试命令 `xp /512b 0x7e00` 验证第二个用户可用区域是空白的。



```
Bochs for Windows - Console
0x000000000000007a0 <hogus+ 416>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000007a8 <hogus+ 424>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000007b0 <hogus+ 432>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000007b8 <hogus+ 440>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000007c0 <hogus+ 448>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000007c8 <hogus+ 456>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000007d0 <hogus+ 464>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000007d8 <hogus+ 472>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000007e0 <hogus+ 480>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000007e8 <hogus+ 488>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000007f0 <hogus+ 496>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000007f8 <hogus+ 504>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
<bochs:12>
```

```
EOS操作系统实验教程_2.1.pdf - Adobe Reader
Bochs for Windows - Console
0x000000000000007fa0 <hogus+ 416>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000007fa8 <hogus+ 424>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000007fb0 <hogus+ 432>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000007fb8 <hogus+ 440>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000007fc0 <hogus+ 448>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000007fc8 <hogus+ 456>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000007fd0 <hogus+ 464>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000007fd8 <hogus+ 472>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000007fe0 <hogus+ 480>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000007fe8 <hogus+ 488>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000007ff0 <hogus+ 496>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x000000000000007ff8 <hogus+ 504>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
<bochs:13>
```

自己设计两个查看内存的调试命令，分别验证这两个用户可用区域的高地址端也是空白的（xp /512b 0x7a00 查看 512b 到 0x7c00，xp /512b 0x9e00 查看 512b 到 0xa0000，可以看出字节码为 0，说明是空白的）

```

Bochs for Windows - Console
0x000000000000c01e8 <bogus+ 488>: 0x60 0xbb 0x00 0xc0 0x8e
0xdb 0xe8 0x64
0x000000000000c01f0 <bogus+ 496>: 0x34 0x61 0x1f 0x07 0x9d
0xcf 0x00 0x00
0x000000000000c01f8 <bogus+ 504>: 0x00 0x04 0x00 0xb8 0xff
0x02 0x01 0x00
<bochs:19> xp /512b 0x7a00
[bochs]:
0x0000000000007a00 <bogus+ 0>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007a08 <bogus+ 8>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007a10 <bogus+ 16>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007a18 <bogus+ 24>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007a20 <bogus+ 32>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007a28 <bogus+ 40>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007a30 <bogus+ 48>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007a38 <bogus+ 56>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007a40 <bogus+ 64>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00

```

试 boot.asm 文件中的汇编代码。按照下面的步骤

```

Bochs for Windows - Console
0x00 0x00 0x00
0x0000000000007bc8 <bogus+ 456>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007bd0 <bogus+ 464>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007bd8 <bogus+ 472>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007be0 <bogus+ 480>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007be8 <bogus+ 488>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007bf0 <bogus+ 496>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007bf8 <bogus+ 504>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
<bochs:20> xp /512b 0x9e00
[bochs]:
0x0000000000009e00 <bogus+ 0>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000009e08 <bogus+ 8>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000009e10 <bogus+ 16>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000009e18 <bogus+ 24>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00

```

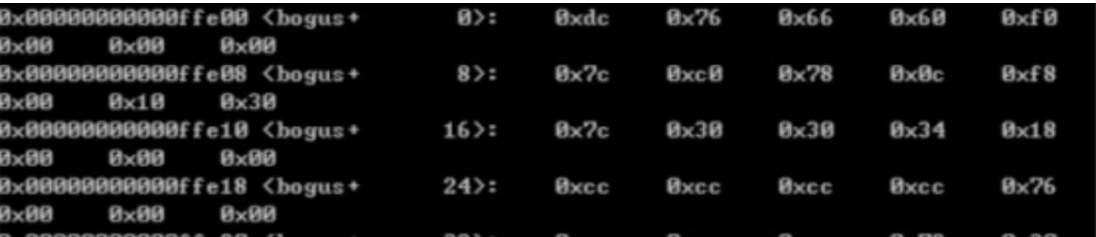
输入调试命令 xp /512b 0xa0000 验证上位内存已经被系统占用。

```

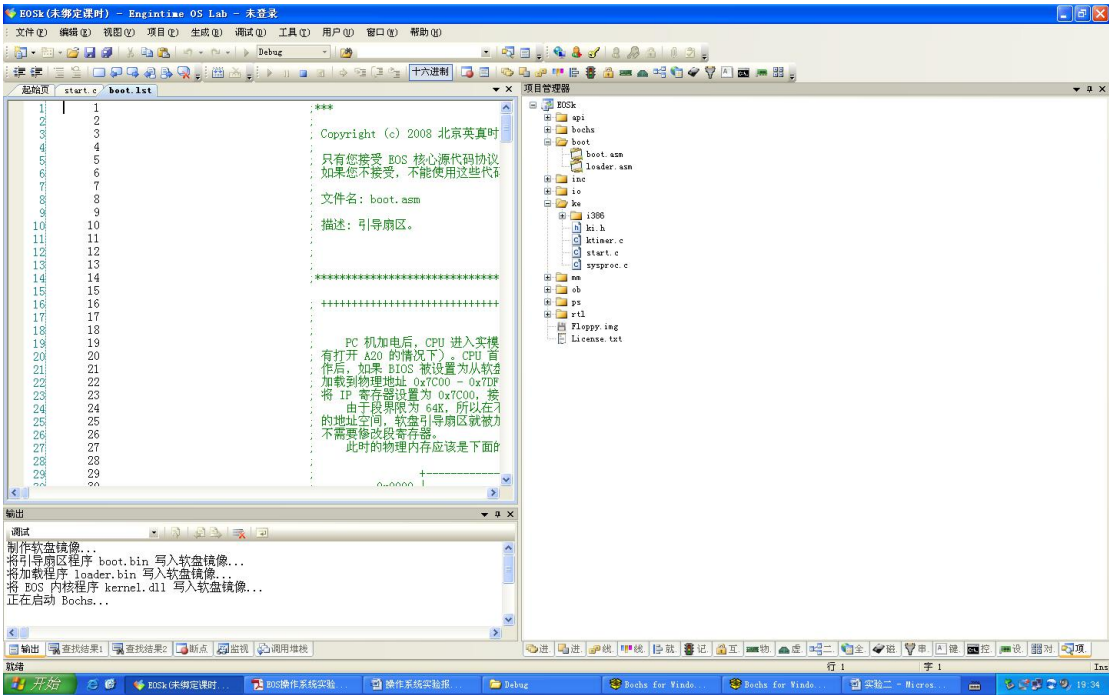
Bochs for Windows - Console
0x00000000000a01a0 <bogus+ 416>: 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff
0x00000000000a01a8 <bogus+ 424>: 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff
0x00000000000a01b0 <bogus+ 432>: 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff
0x00000000000a01b8 <bogus+ 440>: 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff
0x00000000000a01c0 <bogus+ 448>: 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff
0x00000000000a01c8 <bogus+ 456>: 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff
0x00000000000a01d0 <bogus+ 464>: 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff
0x00000000000a01d8 <bogus+ 472>: 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff
0x00000000000a01e0 <bogus+ 480>: 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff
0x00000000000a01e8 <bogus+ 488>: 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff
0x00000000000a01f0 <bogus+ 496>: 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff
0x00000000000a01f8 <bogus+ 504>: 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff
<bochs:15>

```

自己设计一个查看内存的调试命令，验证上位内存的高地址端已被系统占用（xp /512b 0xffe00，查看 0xffe00 之后 512b 空间上字节是否不为 0 ）



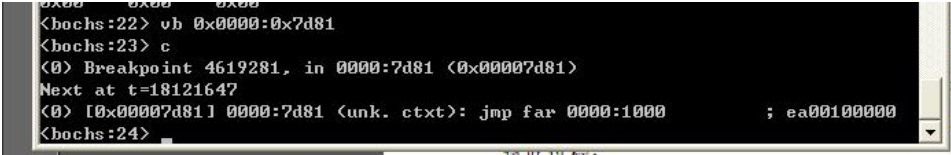
在“项目管理器”窗口中，右键点击“boot”文件夹中的 boot.asm 文件。在弹出的快捷菜单中选择“打开生成的列表文件”，在源代码编辑器中就会打开文件 boot.lst。



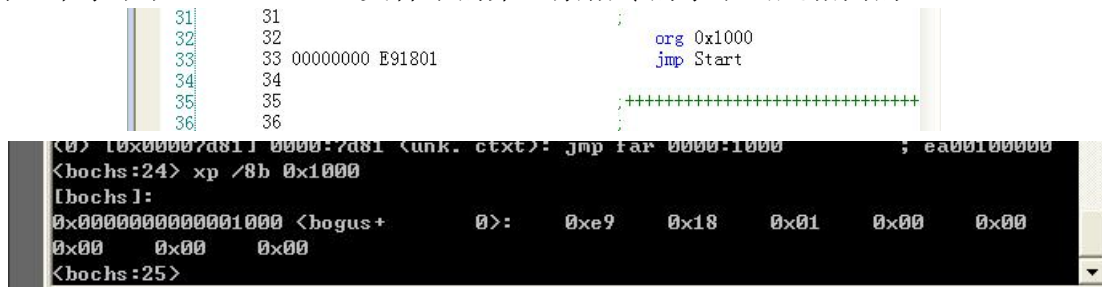
在 boot.lst 中查找到软盘引导扇区程序第一条指令所在的行（第 73 行）

```
72 72 org 0x7C00
73 73 jmp short Start
74 74 nrm
```

输入调试命令 vb 0x0000:0x7d81 添加一个断点。



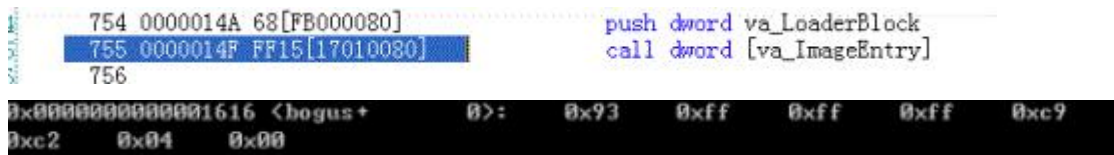
输入调试命令 `xp /8b 0x1000` 查看内存 `0x1000` 处的数据，验证此块内存的前三个字节和 `loader.lst` 文件中的第一条指令的字节码是相同的。



The screenshot shows a debugger window with two panes. The top pane displays assembly code with addresses 31 to 36. Address 31 has the instruction `org 0x1000`, and address 32 has `jmp Start`. The bottom pane shows the command prompt with the command `<bochs:24> xp /8b 0x1000` and its output. The output shows a memory dump starting at `0x0000000000001000` with values `<bogus+ 0>: 0xe9 0x18 0x01 0x00 0x00`. The prompt then changes to `<bochs:25>`.

根据之前记录的 `loader.bin` 文件的大小，自己设计一个查看内存的调试命令，查看内存中 `loader` 程序结束位置的字节码，并与 `loader.lst` 文件中最后指令的字节码比较。

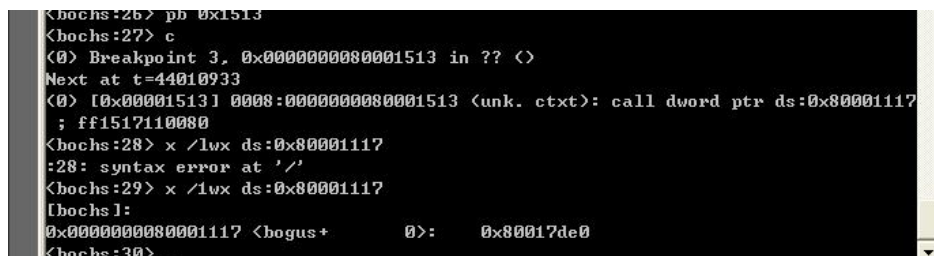
(`1566b` 十六进制为 `61e`，起始 `0x1000+61e-8b=0x1616`，最终才能查看最后八位，所以命令为 `xp /8b 0x1616`)



The screenshot shows a debugger window. The top pane displays assembly code with addresses 754 to 756. Address 754 has the instruction `push dword va_LoaderBlock`, and address 755 has `call dword [va_ImageEntry]`. The bottom pane shows a memory dump starting at `0x0000000000001616` with values `<bogus+ 0>: 0x93 0xff 0xff 0xff 0xc9 0xc2 0x04 0x00`.

3.2.4. 调试加载程序

使用添加物理地址断点的调试命令 `pb 0x1513` 添加一个断点，输入调试命令 `c` 继续执行，在刚刚添加的断点处中断。在 `Console` 窗口中显示要执行的下一条指令，使用查看虚拟内存的调试命令 `x /lwx ds:0x80001117` 查看内存中保存的 32 位函数入口地址。



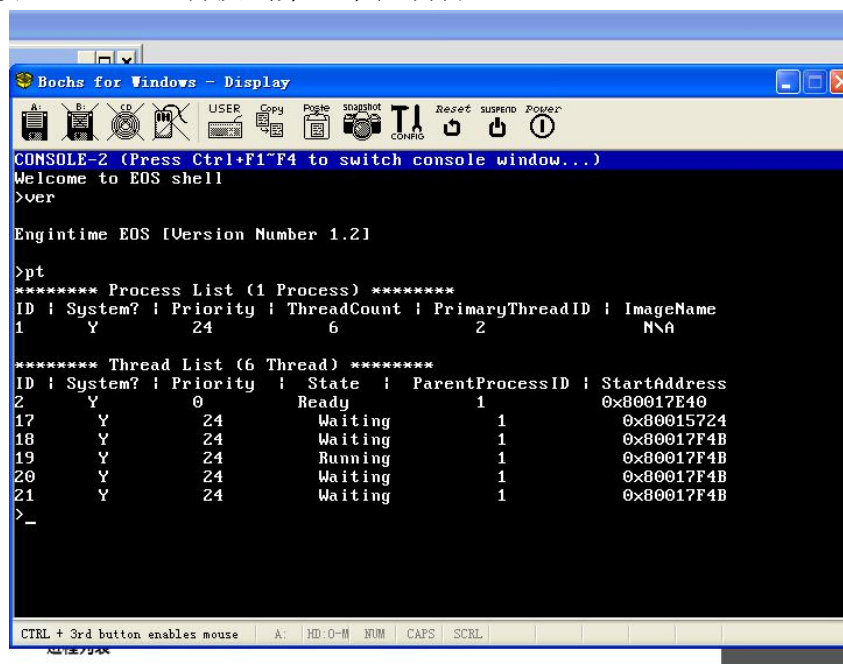
The screenshot shows a debugger window. The top pane displays the command prompt with the command `<bochs:26> pb 0x1513` and its output. The output shows the breakpoint setup: `<bochs:27> c`, `<0> Breakpoint 3, 0x0000000000001513 in ?? <>`, `Next at t=44010933`, `<0> [0x00001513] 0008:0000000000001513 <unk. ctxt>: call dword ptr ds:0x80001117`, `; ff1517110080`. The bottom pane shows the command prompt with the command `<bochs:28> x /lwx ds:0x80001117` and its output. The output shows a syntax error: `:28: syntax error at '/'`. The prompt then changes to `<bochs:29> x /lwx ds:0x80001117` and the output shows the memory dump: `[bochs]: 0x0000000000001117 <bogus+ 0>: 0x80017de0`. The prompt then changes to `<bochs:30>`.

3.2.5. 调试内核

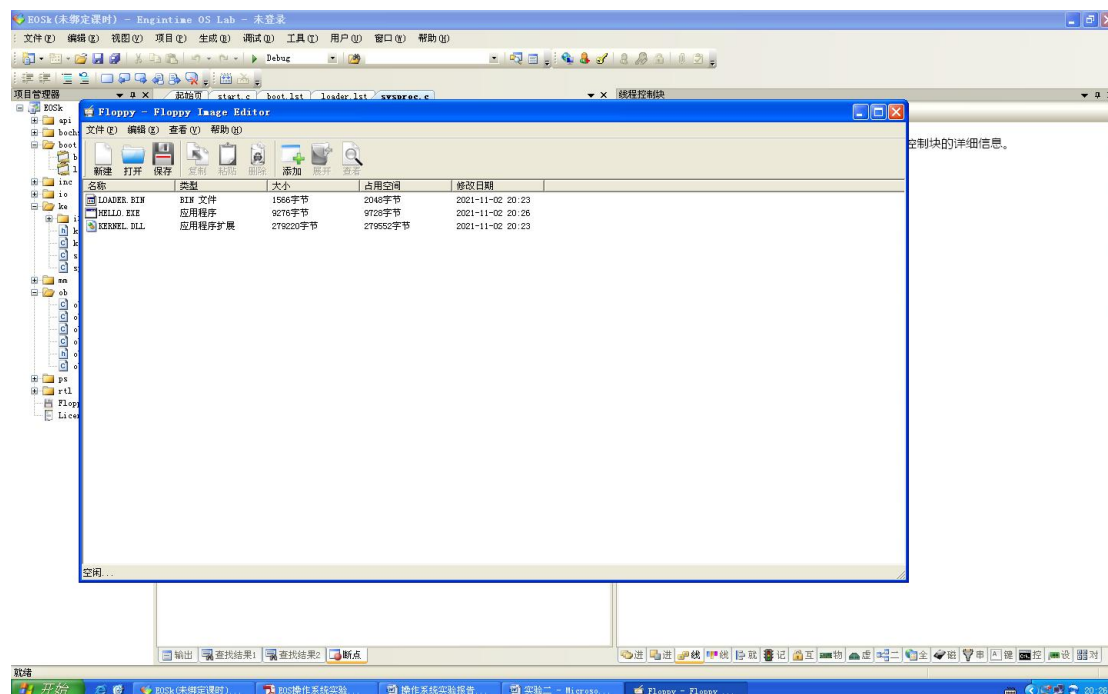
在“项目管理器”窗口中，右键点击项目节点，在弹出的快捷菜单中选择“属性”。在弹出的“属性页”对话框右侧的属性列表中找到“远程目标机”属性，将此属性值修改为“Bochs GDB stub”。

在 `KiSystemStartup` 函数中的代码行（第 52 行）`KiInitializePic()`；添加一个断点。

按 Ctrl+F2，切换到第二个控制台



查看当一个 EOS 应用程序正在运行时的进程和线程信息，在 OS Lab “项目管理器” 窗口中双击 Floppy.img 文件，使用 FloppyImageEditor 工具打开此软盘镜像文件，将本实验文件夹中的 Hello.exe 文件拖动到 FloppyImageEditor 工具窗口的文件列表中释放。



[illegible]

Bochs for Windows - Display

USER: root

Beetle: superio Power

CONSOLE-2 (Press Ctrl+F1F4 to switch console window...)

Welcome to EOS shell

pt

***** Process List (1 Process) *****

ID	System?	Priority	ThreadCount	PrimaryThreadID	ImageName
1	Y	24	6	2	N/A

***** Thread List (6 Thread) *****

ID	System?	Priority	State	ParentProcessID	StartAddress
2	Y	0	Ready	1	0x80017F40
17	Y	24	Waiting	1	0x80015724
18	Y	24	Waiting	1	0x80017F4B
19	Y	24	Running	1	0x80017F4B
20	Y	24	Waiting	1	0x80017F4B
21	Y	24	Waiting	1	0x80017F4B

CTRL + 3rd button enables mouse

Bochs for Windows - Display

Process List (1 Process)

Thread List (6 Thread)

Process ID : StartAddress

1 : 0x80017E40

1 : 0x80015724

1 : 0x80017F4B

1 : 0x80017F4B

1 : 0x80017F4B

1 : 0x80017F4B

1 : 0x80017F4B

线程和线程的信息

信息。接下来，读者可以按照下面的步骤查看

结束之前的调试。

使用FloppyImageEditor工具打开此软盘镜像

文件夾。可以在本书前部部分找到“学生包”的

下载地址。

5. 将本实验文件夹中的Hello.exe文件拖动到FloppyImageEditor工具窗口的文件列表中释放，Hello.exe文件即被添加到软盘镜像文件中。Hello.exe是一个EOS应用程序，其源代码可以参见本实验文件夹中的Hello.c源文件。
6. 在FloppyImageEditor中选择“文件”菜单中的“保存”后关闭FloppyImageEditor。
7. 按 F5 启动调试。
8. 待 EOS 启动完毕，在 EOS 控制台中输入命令“hello”后按回车。此时在软盘中的 EOS 应用程序 Hello.exe 就会开始运行。
9. 迅速按 Ctrl+F2 切换到控制台 2，并输入命令“pt”后按回车。输出的进程和线程信息如图 10-7 所示。

4. 实验的思考与问题分析

4.1. 为什么 EOS 操作系统从软盘启动时要使用 `boot.bin` 和 `loader.bin` 两个程序？使用一个可以吗？它们各自的主要功能是什么？如果将 `loader.bin` 的功能移动到 `boot.bin` 文件中，则 `boot.bin` 文件的大小是否仍然能保持小于 512 字节？

答：`Boot.bin` 和 `loader.bin` 分别是 `boot.asm` 和 `loader.asm` 汇编生成的，前者用于引导软盘，后者用于加载程序，启动执行 EOS 操作系统时，会将 `boot.bin`、`loader.bin` 和 `kernel.dll` 文件写入软盘镜像文件中，然后让虚拟机来执行软盘中的 EOS 操作系统，所以不能只使用一个。不能保持，将 `loader.bin` 的功能移动到 `boot.bin` 文件中，会导致文件体积大于 512 字节。

4.2. 软盘引导扇区加载完毕后内存中有两个用户可用的区域，为什么软盘引导扇区程序选择将 `loader.bin` 加载到第一个可用区域的 `0x1000` 处呢？这样做有什么好处？这样做会对 `loader.bin` 文件的大小有哪些限制。

答：第一个用户可用区域是低地址区，空间大小适合加载 `loader.bin` 小文件。

优点：从低地址开始便于查找；节约空间资源。

限制：低地址区空间小，所以 `loader.bin` 文件大小不能大于 1c00k。

5. 总结和感想体会

通过这次实验了解了操作系统的启动过程和 EOS 启动后状态和行为，知道了 `boot` 程序和 `loader` 程序的执行情况，在启动之后命令调试的过程中，学会了常用的调试命令，会对内存中的情况进行查看，明白了内存中的加载情况。通过对程序代码进行打断点，逐步跟踪调试，明白了内核初始化的流程。